OPTIMIZING MEMORY ACCESSES FOR NETWORK APPLICATIONS
USING INDEXED REGISTER FILES

Background

Field

[0001]   The embodiments relate to high-speed network devices, and more particularly to optimizing memory access for high-speed network devices.

Description of the Related Art

[0002]   Synchronous optical network (SONET) is a standard for optical telecommunications transport formulated by the Exchange Carriers Standards Association (ECSA) for the American National Standards Institute (ANSI), which sets industry standards in the U.S. for telecommunications and other industries. Network processors (NP) are emerging as a core element of network devices, such as high-speed communication routers. NPs are designed specifically for network processing applications.

[0003]   The unique challenge of network processing is to guarantee and sustain throughput for the worst-case traffic. For instance, the case of the optical level OC-192 (10 Gigabits/sec) POS (Packet over SONET) packet processing presents significant processing and throughput challenges. It requires a throughput of 28 million packets per second or service time of 4.57 microseconds per packet for processing in the worst case. The latency for a single external memory access is much larger than the worst-case service time.

[0004]   Therefore, modern network processors usually have a highly

1

parallel architecture with non-uniform memory hierarchy. Network processors can consist of multiple microengines (*MEs*, or programmable processors with packet processing capability) running in parallel. Each ME has its own local memory (*LM*), for example registers.

[0005] Various constraints may be applied to accessing register files, which complicates the management of the register files. For example, a local memory in a NP can be addressed using a *BASE-OFFSET* word address. The *BASE* value is stored in a specific base-address register, and there is 3-cycle latency between writing the base-address register when its value changes.

[0006] The *OFFSET* is a constant from 0 to 15. The final address in the *BASE-OFFSET* mode, however, is computed using a logical *OR* operation (i.e., *BASE | OFFSET*). Therefore, to support C pointer arithmetic, e.g., *pointer + offset*, using the *BASE-OFFSET* mode of local memory where *BASE = pointer* and *OFFSET = offset*, proper alignment of *BASE* has to be ensured such that the condition in **Fig. 1** holds. Otherwise, to access that address, the base-register has to be set to *pointer + offset*, and the *OFFSET* is set to 0. **Fig. 1** illustrates the alignment requirement of the BASE-OFFSET addressing mode of the local memory.

CONTENTS OF THE INVENTION

Problem to be solved

[0007]   Current network processors (NP) have latency between writing the base-address register and when its value changes. Further latency is added when accessing external memory to the NP. Therefore, the problem is how to reduce latency with memory accesses.

Solutions

[0008]   In order to improve performance for network applications, one embodiment includes an optimizing compiler to optimize and minimize external memory accesses using the local memory (i.e., indexed register files), and minimizes the initializations of the base-address register for efficient local memory accesses.

[0009]   One embodiment migrates external memory objects (e.g., variables) to the local memory (i.e., indexed register files), and optimizes the accesses to the local memory by determining alignment of the migrated objects; and eliminating redundant initialization code of the objects.

[0010]   The advantages of the embodied solutions is that objects that are accessed from external memory are now accessed through local memory to a network processor (e.g., indexed registers) and the latency from writing the base-address register when its value changes is reduced as redundant initializations are eliminated.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011]   The embodiments are illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0012]   Fig. 1 illustrates the alignment requirement of the BASE-OFFSET addressing mode of the local memory;

[0013]   Fig. 2 illustrates a block diagram of an embodiment;

[0014]   Fig. 3 illustrates a block diagram of object migration;

[0015]   Fig. 4A-B illustrates original object and code sequences in external memory;

[0016]   Fig. 5A-B illustrates objects migrated to local memory and access code sequence without alignment adjustment;

[0017]   Fig. 6 illustrates a block diagram for determining alignment of migrated objects;

[0018]   Fig. 7 illustrates pseudo-code for determining and setting the minimum alignment needed for each object;

[0019]   Fig. 8A-B illustrates objects migrated to local memory and access code sequence with alignment adjusted;

[0020]   Fig. 9A-B illustrates objects migrated to local memory and access code sequence with alignment adjusted and redundant initializations eliminated;

[0021]   Fig. 10 illustrates an embodiment of a processing device;

[0022]   Fig. 11 illustrates an optimizer system for network processors; and

[0023]   Fig. 12 illustrates a compiler of the embodiment illustrated in Fig. 10.

DETAILED DESCRIPTION

[0024]   The embodiments discussed herein generally relate to optimization of local memory accessing and latency reduction for network processors. Referring to the figures, exemplary embodiments will now be described. The exemplary embodiments are provided to illustrate the embodiments and should not be construed as limiting the scope of the embodiments.

[0025]   Fig. 2 illustrates a general block diagram of an embodiment for optimizing an executable. In this embodiment external memory accesses are optimized using indexed register files, by efficiently migrating external memory variables to the local memory (i.e., indexed register files), and minimizing the initializations of the base-address register of the local memory. In block 1 objects that change values (e.g., variables) are migrated from an external memory, such as a main memory, such as random-access memory (RAM), static random access memory (SRAM), dynamic random access memory (DRAM), synchronous DRAM (SDRAM), read-only memory (ROM), etc., of a network processor (NP) to local memory of the NP, such as indexed registers. In block 2, alignment of the migrated objects is determined. And, in block 3 redundant initialization code of the objects is eliminated. Fig. 3 illustrates block 1 in further detail.

[0026]   Fig. 3 illustrates a block diagram of object migration. In general, the blocks in Fig. 3 cover determining whether each object of the plurality of objects are accessible from a plurality of processors; determining an equivalence set of aliased objects in the plurality of objects; determining objects of the plurality of objects eligible for migration; changing residence of the objects determined to be eligible for migration; and changing accesses of the objects having their residence changed.

[0027]   As illustrated in Fig. 2, first variables are migrated from external

5

memory to local memory (i.e., indexed registers). In one embodiment, the eligible objects (i.e., variables) in external memory are migrated to the local memory. That is, the residences of those variables are changed to the local memory, and the accesses to those variables are changed correspondingly.

[0028]   Since local memory resides in each NP and the local memory in one processor cannot be shared with another processor, variables that are accessed by multiple processors are not migrated to local memory. In block 1.1 it is determined whether a variable is accessed by multiple processors through escape analysis. In one embodiment, escape analysis determines whether an object (i.e., variable) is accessed by more than one processor. Consequently, variables in external memory can be migrated to indexed register files for fast accesses, no matter whether they are accessed using constant addresses or pointers (i.e., non-constant addresses).

[0029]   In block 1.2 an equivalence set of aliased variables are computed through points-to analysis. That is, variables that could possibly be accessed by one instruction belong to the same equivalence set. If one variable in an equivalence set cannot be migrated to local memory, none of those variables in the same equivalence set can be migrated to local memory. In one embodiment the total size of variables should not exceed the available local memory size. With the above constraints and the equivalence set, variables that are eligible for migration are computed in block 1.3.

[0030]   In block 1.4, the residence of eligible variables is changed from external memory to local memory. In block 1.5, accesses of those variables whose residence were changed is changed.

[0031]   For example, suppose there are three variables A, B, C in an external memory (e.g. SRAM) whose original alignment and size are illustrated in Fig. 4A. The access order is illustrated in Fig. 4B. For the access order of A, B, and C as illustrated in Fig. 4B, several I/O operations to the external

6

memory are needed.

[0032]   Fig. 5A-B illustrate data migrated to local memory and access code sequence without alignment adjustment. For ease of discussion, suppose A, B, and C satisfy the migration condition, after migrating these objects to local memory, accesses of these objects with local memory base address register initialization code inserted are illustrated in Fig. 5B. Without further optimization, none of the accesses can share the base address value because the base address and offset value do not satisfy the alignment requirement of the BASE-OFFSET mode.

[0033]   Fig. 6 illustrates a block diagram of determining the alignment of migrated variables through a forward disjunctive dataflow analysis. In one embodiment, the alignments of the migrated objects are adjusted properly, such that the sharing of the base address register is maximized between the accesses to the local memory. In one embodiment, the minimum alignments required for objects in local memory to maximize sharing of base registers and to reduce padding between variables are determined. That is, the alignment of the object in local memory is determined such that, any smaller alignment causes less sharing of the base-address register, and any alignment larger than this value does not cause more sharing of the base-address address.

[0034]   Block 2.1 uses a forward disjunctive dataflow analysis to compute the offset value pairs with a common base address. The dataflow analysis uses a simplified flow graph, i.e., those instructions that do not contain any accesses to migrated objects are purged off and each flow node consists of only one instruction.

[0035]　　　　In the simplified flow graph, flow nodes and instructions are the same. In one embodiment, it is assumed that each instruction contains, at most, one local memory access, and the address of the access is expressed in the form of *base address* + *constant offset*. The dataflow equations for each instruction *i* is shown below.

$$
\begin{cases}
GEN[i] = \{L \mid L \text{ is the local memory base addess used in instruction } i\} \\
KILL[i] = \{L \mid L \text{ is the local memory base address not used in instruction } i\} \\
IN[i] = \bigcup_{p \in Pred(i)} OUT[p] \\
OUT[i] = GEN[i] \bigcup (IN[i] - KILL[i])
\end{cases}
$$

[0036]　　　　The forward disjunctive dataflow analysis is iterated until both IN and OUT are converged. For the example of sequential accesses illustrated in **Fig. 5B**, the values of GEN and KILL for each local memory access are as follows:

```
GEN(1) = {A[i][0]}    KILL(1) = {B[i][0],C[i][0]}
GEN(2) = {A[i][0]}    KILL(2) = {B[i][0],C[i][0]}
GEN(3) = {B[i][0]}    KILL(3) = {A[i][0],C[i][0]}
GEN(4) = {B[i][0]}    KILL(4) = {A[i][0],C[i][0]}
GEN(5) = {B[i][0]}    KILL(5) = {A[i][0],C[i][0]}
GEN(6) = {B[i][0]}    KILL(6) = {A[i][0],C[i][0]}
GEN(7) = {C[i][0]}    KILL(7) = {A[i][0],B[i][0]}
GEN(8) = {C[i][0]}    KILL(8) = {A[i][0],B[i][0]}
```

[0037]　　　　The final values of IN and OUT are as follows:

```
IN(1) = {}          OUT(1) = {A[i][0]}
IN(2) = {A[i][0]}   OUT(2) = {A[i][0]}
IN(3) = {A[i][0]}   OUT(3) = {B[i][0]}
IN(4) = {B[i][0]}   OUT(4) = {B[i][0]}
IN(5) = {B[i][0]}   OUT(5) = {B[i][0]}
IN(6) = {B[i][0]}   OUT(6) = {B[i][0]}
IN(7) = {B[i][0]}   OUT(7) = {C[i][0]}
IN(8) = {C[i][0]}   OUT(8) = {C[i][0]}
```

[0038]   In one embodiment, each base address in GEN[i] ∩ IN [i] is used

by two consecutive local memory accesses to the same object, with possibly

different (constant) offsets. In one embodiment, if the base address and one of

the constant offsets do not satisfy the requirement in **Fig. 1**, the sharing of the

base address is not possible. In this case, one embodiment of an optimizer (e.g.,

compiler) can enlarge the alignment to the objects so that the base address and

offset values can meet the alignment requirement.

[0039]   In one embodiment, the pair of two different offset values (*offset

value pair*) of the two consecutive local memory accesses that use the same base

address can be computed during the dataflow iteration. That is, when

calculating the IN set for flow node *i*, if GEN[i] ∩ IN [i] is found not to be

empty, the different offset values of the current and previous local memory

accesses (that use the same base address) are recorded as a pair of offset values

(associated with the base address). In the above example, the list of offset value

pairs associated with the base address is shown below.

```
A[i][0] -> {(0,4)}
B[i][0] -> {(0,4), (4,8), (8,12)}
C[i][0] -> {(0,4)}
```

[0040]   For each base address, assume *VAR* is a variable accessed by this

base address and its size is *SIZE*; then the upper bound of the alignment to be

attempted for *VAR*, or *MAX_ALIGN(VAR)*, can be determined as follows. Here

the *MAX_ALIGN* is the width (in bytes) of the *OFFSET* in the BASE-OFFSET

addressing mode (for instance, 64 for the local memory in a NP).

$$MAX\_ALIGN(VAR) = \min\left(MAX\_ALIGN, \ 2^\wedge\lceil \log_2 SIZE \rceil \right)$$

[0041]   Fig. 7 illustrates pseudo code of block 2.2 that determines and sets

the minimum alignment needed for each object (i.e., variable) in local memory.

In one embodiment, with the pseudo-code in **Fig. 7**, the alignment of the

objects can be properly set so that two consecutive accesses of the same

variable can use the same base address value, thus reducing the number of

base address register initialization instructions. According to the definition of

the minimum alignment and the pseudo-code illustrated in **Fig. 7**, the

minimum alignments of A, B, C are set to 8, 16 and 8 bytes, respectively. With

the adjusted alignment, the data in local memory and access code sequence are

illustrated in **Fig. 8A-B**.

[0042]   The result of block 2.2 (illustrated in **Fig. 6**) could contain some
redundant initializations of the base address register. In one embodiment,
those redundant initializations are eliminated using any existing (partial)
redundancy elimination algorithms. After the redundant initialization code
elimination, the data in local memory and the access code sequence is
illustrated in **Fig. 9A-B**, in which the initialization instructions are greatly
reduced.

[0043]   **Fig. 10** illustrates a processing device 1000. Processing device
1000 includes a processing device (e.g., a NP) 1020 connected to external
memory 1010 (e.g., SDRAM). Processing device 1000 further includes
optimizer 1040 that is coupled to processor 1030. Further included are indexed
registers 1050 coupled to processing device 1020. As illustrated, objects 1060
currently reside in indexed registers 1060 after being migrated from external
memory 1010. Optimizer 1040 minimizes external memory 1010 accesses by

migrating objects 1060 (e.g., variables) in external memory to indexed registers 1050 (e.g., the local memory on processing device 1020). That is, changing the residence of those objects to indexed registers 1050 and the accesses to those objects. Consequently, external memory 1010 access latency is minimized for network applications. In one embodiment, processing device 1000 is a high-speed networking router. On another embodiment, multiple processing devices 1020 are included in processing device 1000.

[0044]   Fig. 11 illustrates an optimizer system 1100. Optimizer system 1100 includes a processing device including processor 1110 coupled to memory 1120, display 1140 and optimizer 1130. In one embodiment, optimizer 1130 is a compiler. Display 1140 can be any known type of display device, such as liquid crystal display (LCD), cathode ray tube (CRT), flat screen technology, projection display, etc. Optimizer system further is coupled to processing device 1000 that does not include optimizer 1040. In one embodiment, optimizer system 1100 is removably coupled to processing device 1000. That is, optimizer system can be coupled to processing device 1000 with a cable, through a network, wireless connection, etc. In one embodiment, optimizer 1130 optimizes processing device 1000 by migrating objects from external memory 1010 to local memory (i.e., indexed registers) 1050 using escape analysis and points-to analysis in a straightforward way. Computation of the minimal alignments of objects (i.e., variables) in local memory 1050 to maximize sharing of base registers and reduce padding between objects through a forward disjunctive dataflow analysis which takes the code proximity into account. Optimizer 1130 further minimizes costly initialization operations through an existing redundancy elimination algorithm.

[0045]   Fig. 12 illustrates components of optimizer 1130. In one embodiment optimizer 1130 includes first determiner 1210 that determines whether each object (e.g., variable) of a plurality of objects are accessible from more than one processor in a network device. Second determiner 1220

11

determines an equivalence set of aliased objects in the plurality of objects. Third determiner 1230 determines objects of the plurality of objects eligible for migration. Migrator 1240 changes residence of the objects determined to be eligible for migration. Accessor 1250 changes accesses of the objects having their residence changed. Optimizer 1130 maximizes the sharing of the base-address registers and minimizes the padding between objects in indexed registers 1050 by properly adjusting the alignments of those objects. Together with redundancy elimination, optimizer 1130 minimizes the initializations of the base-address registers.

[0046]   Embodiments of the present disclosure described herein may be implemented in circuitry, which includes hardwired circuitry, digital circuitry, analog circuitry, programmable circuitry, and so forth. These embodiments may also be implemented in computer programs. Such computer programs may be coded in a high level procedural or object oriented programming language. The program(s), however, can be implemented in assembly or machine language if desired. The language may be compiled or interpreted. Additionally, these techniques may be used in a wide variety of networking environments. Such computer programs may be stored on a storage media or device (e.g., hard disk drive, floppy disk drive, read only memory (ROM), CD-ROM device, flash memory device, digital versatile disk (DVD), or other storage device) readable by a general or special purpose programmable processing system, for configuring and operating the processing system when the storage media or device is read by the processing system to perform the procedures described herein. Embodiments of the disclosure may also be considered to be implemented as a machine-readable or machine recordable storage medium, configured for use with a processing system, where the storage medium so configured causes the processing system to operate in a specific and predefined manner to perform the functions described herein.

[0047]   While certain exemplary embodiments have been described and

12

shown in the accompanying drawings, it is to be understood that such embodiments are merely illustrative of and not restrictive on the broad invention, and that this invention not be limited to the specific constructions and arrangements shown and described, since various other modifications may occur to those ordinarily skilled in the art.

[0048]   Reference in the specification to "an embodiment," "one embodiment," "some embodiments," or "other embodiments" means that a particular feature, structure, or characteristic described in connection with the embodiments is included in at least some embodiments, but not necessarily all embodiments. The various appearances "an embodiment," "one embodiment," or "some embodiments" are not necessarily all referring to the same embodiments. If the specification states a component, feature, structure, or characteristic "may", "might", or "could" be included, that particular component, feature, structure, or characteristic is not required to be included. If the specification or claim refers to "a" or "an" element, that does not mean there is only one of the element. If the specification or claims refer to "an additional" element, that does not preclude there being more than one of the additional element.